USING AUTOMATION TO IMPROVE THE FLIGHT SOFTWARE TESTING PROCESS

James R. O'Donnell, Jr., Ph.D.[1], Wendy M. Morgenstern[1], Maureen O. Bartholomew[2]
NASA Goddard Space Flight Center
Greenbelt, MD 20771 USA

One of the critical phases in the development of a spacecraft attitude control system (ACS) is the testing of its flight software. The testing (and test verification) of ACS flight software requires a mix of skills involving software, knowledge of attitude control and attitude control hardware, data manipulation, and analysis. The process of analyzing and verifying flight software test results often creates a bottleneck which dictates the speed at which flight software verification can be conducted. In the development of the Microwave Anisotropy Probe (MAP) spacecraft ACS subsystem, an integrated design environment was used that included a MAP high fidelity (HiFi) simulation, a central database of spacecraft parameters, a script language for numeric and string processing, and plotting capability. In this integrated environment, it was possible to automate many of the steps involved in flight software testing, making the entire process more efficient and thorough than on previous missions. In this paper, we will compare the testing process used on MAP to that used on other missions. The software tools that were developed to automate testing and test verification will be discussed, including the ability to import and process test data, synchronize test data and automatically generate HiFi script files used for test verification, and an automated capability for generating comparison plots. A summary of the benefits of applying these test methods on MAP will be given. Finally, the paper will conclude with a discussion of re-use of the tools and techniques presented, and the ongoing effort to apply them to flight software testing of the Triana spacecraft ACS subsystem.

INTRODUCTION

The development of the attitude control system (ACS) for the Microwave Anisotropy Probe (MAP) spacecraft (see Figure 1) included a number of firsts for the Goddard Space Flight Center. It was the first time that the MatrixX integrated simulation, analysis, and design toolkit was used at Goddard. Of greater note, it was the first time that automatically generated flight software, generated from the MAP high fidelity (HiFi) simulation using the AutoCode feature of the MatrixX toolkit, was used here. A number of papers (ref. 1, 2, 3) have been written about the use of the MatrixX toolkit on the MAP project, as well as on the special considerations that needed to be taken into account when testing automatically generated flight software.

During the development and testing of the MAP ACS subsystem, a number of tools were developed to automate portions of the ACS flight software test process. This set of MAP test tools, combined with some of the other integrated features of the MAP project, greatly increased the efficiency and thoroughness of the flight software testing. In the remainder of this paper, these tools will be discussed, along with a discussion of their application to the Triana spacecraft.

---

[1] Flight Dynamics Analysis Branch, Code 572
[2] Flight Software Branch, Code 582

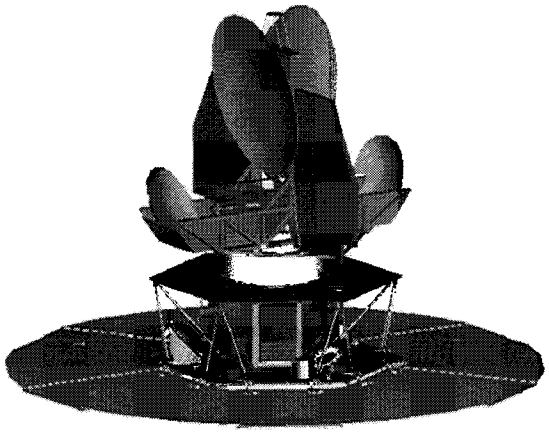

Figure 1: The MAP Spacecraft

FLIGHT SOFTWARE TEST PROCESS

On the MAP project, as with many past and present missions, the ACS flight software test process was divided up into a number of steps. At the lowest level, as the flight software developers wrote each piece of code, tests were performed at the unit test level. Once the flight software for the spacecraft was integrated onto the spacecraft or test facility, software testing was done first at the build test and then the acceptance test levels. During build testing, testers verified the correct operation of each specific flight software function. By and large, each of these tests was driven by a specified requirement of the software, and testing was used to verify that the requirement was met. At the acceptance test level, more realistic tests were run that verified that the spacecraft would operate in both nominal conditions and in the face of anomalies. These acceptance tests sought to cover the full lifetime of the spacecraft, from launch and early orbit operations to orbit maneuvers to the spacecraft science mode. Finally, in addition to the build and acceptance test levels, the MAP test team performed a number of sets of regression testing; this was an abbreviated set of build and acceptance tests performed after late additions or changes to the flight software.

For MAP, the test facility mentioned above was known as FlatSat. It consisted of engineering test units of the spacecraft main processor, attitude control electronics, and several other electronics boxes. The spacecraft's sensors and actuators, as well as the dynamics and other environmental modeling, was done using a hybrid dynamic simulator (HDS). The MAP HDS had very high fidelity, and it was possible to do very realistic testing with it on FlatSat.

Process Steps

The basic flight software test flow began with developing test scenarios. These scenarios defined the initial conditions of the test as well as the flow of the test, including mode transitions, telemetry verification and failure conditions. From these scenarios, test procedures were developed, then executed on the test environment. The test environment included a ground system that was used to send commands and receive telemetry from the flight system. It also included the flight system comprised of hardware and flight software. Ground support equipment such as the HDS was another critical component of the test environment. Once the tests were executed, the results needed to be analyzed in order to verify that the flight software met the test objectives. In general, the test results needed to be plotted for the analysis.

In parallel, the test scenario was duplicated in the high fidelity (HiFi) simulation, producing results which were used to compare with the flight software test output results from the FlatSat test environment. The HiFi results were the "truth" used to determine if the performance of the flight system was adequate.

There were several aspects of flight software testing critical to its timely success. First, the ability to replicate the initial conditions and test flow in the HiFi is extremely important since the results of the flight software test were compared with the HiFi results. Second, the flight software, the HDS, and the HiFi need to be consistent. There were many variables such as scale factors, biases, and alignments that existed in two or more of these systems and must be identical in each. Third, the ability to plot and analyze the results with ease was crucial.

"Old vs New" Comparison

In order to appreciate the benefits of the automation that was done on MAP, it is important to understand the process and bottlenecks on previous projects. In order to illustrate this, the testing process used for past missions, such as the Rossi X-Ray Timing Explorer (XTE), will be contrasted with that of MAP. It should be noted first, though, that XTE has been a very successful mission; the fact that bottlenecks or possible areas for improvement in their flight software testing process are identified below in no way diminishes from that fact.

XTE Flight Software Test Process

Figure 2 shows the flight software test process used for XTE. As denoted by the grey shaded arrows, there were many steps in this process that were manual and, therefore, prone to errors.

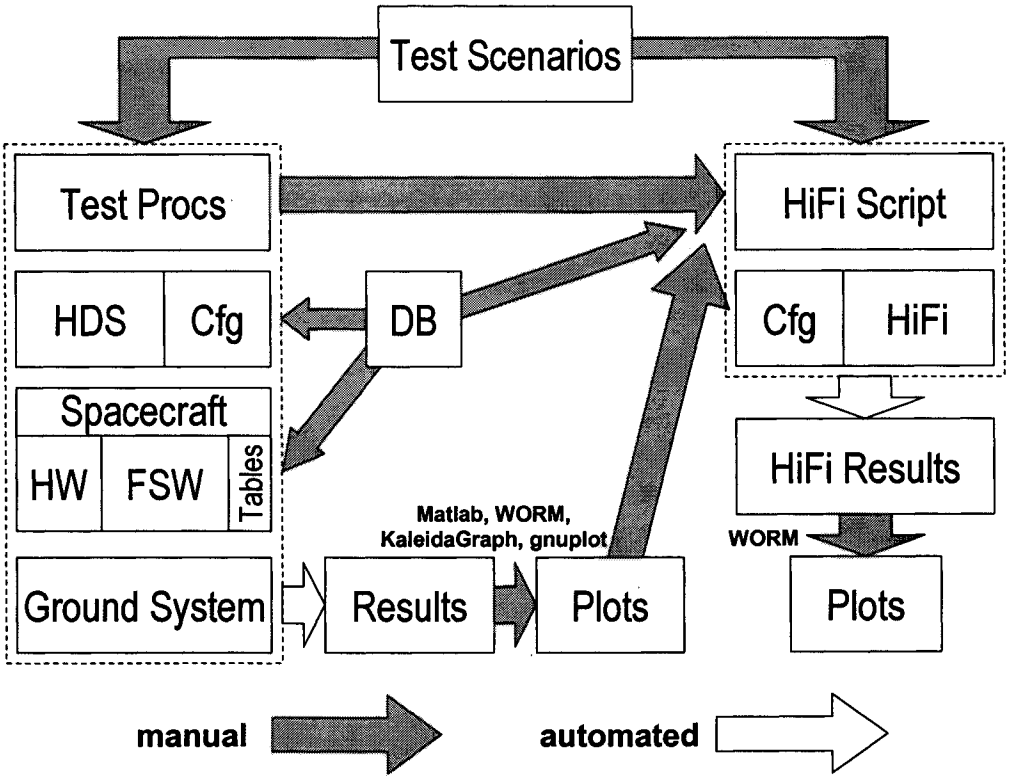

Figure 2: "Old" Flight Software Test Process Includes Many Manual Steps

On XTE, flight software test procedures were developed on the ground system. The initial conditions for these tests were then given to an ACS analyst to replicate the initial conditions in HiFi, which were used to define the expected test results. The flight software test procedures were executed in the flight software lab. The results were plotted using the test author's favorite plotting package, which was usually different from any of the other tester's favorite plotting package. The resulting plots from both the HiFi and the flight software test execution, plotted separately, were then held side to side and visually compared. Another critical piece to the flow was that there existed a plethora of variables that needed to be consistent between the HiFi, HDS, and flight software. On XTE, these variables were maintained in a spreadsheet. The variables were then manually entered into the HiFi, HDS, and flight software.

The pitfalls that the XTE process suffered were in each of those manual processes listed above, namely replicating the initial conditions, plotting of the results, and maintaining consistency between all of the variables. Especially in the infancy of the XTE flight software test program, it was not unusual to go through several iterations of a particular test, trying to match the initial conditions and variables. Inconsistencies would be discovered in the plot results review which would result in a discovery that, for example, the Kalman filter was not enabled in the flight software test but was enabled in the HiFi version of the test, creating different results. Sometimes the inconsistencies were more subtle, such as a number swap of one of the digits of a variable (e.g., flight software gyro bias = 3.0234, HiFi gyro bias = 3.0243). Time consuming iterations were also made in the plotting process. When comparing the flight software test plots with the HiFi plots, consistency in scaling and units are extremely important. Many times, the flight software plots would have to be recreated in order to scale the plots with enough detail to verify the results with HiFi or to change the units to facilitate comparison.

Finally, the separate nature of the flight software testing and HiFi verification paths above created a potential slowdown in the process. The HiFi verification runs could not be created except by a member of the ACS analysis team familiar with the HiFi, and could also not be done with any confidence until the flight software test results were viewed. The limited number of analysts, as well as the fact that the analyst would need to interact with the tester on multiple occasions for each test, frequently resulted in a bottleneck waiting for tests to be verified.

MAP Flight Software Test Process

On MAP, many of the manual processes were automated, streamlining the testing process. Figure 3 illustrates the flight software test flow that MAP followed, highlighting those areas that were automated. Unlike XTE, MAP's flight software testing process had tools that dissected a flight software test procedure, automatically producing a HiFi script that replicated the flight software test procedure flow. In addition, MAP had a centralized database that defined and linked all of the variables used in HiFi, HDS, and the flight software. With the press of a button, the database generated the source files for each of these systems which guaranteed consistency. Finally, the plotting process was streamlined since the MAP tools defined a standard set of plots that were used to plot the flight software, HiFi, and HDS test results on one plot.

In the next two sections of this paper, the specific tools that were developed to perform each of the tasks shown in Figure 3 will be discussed. Additionally, the other assumptions necessary for the correct operation of these tools will be shown; each of these assumptions involved a

specific standard way of writing procedures or formatting output that was established to improve the MAP testing flow.

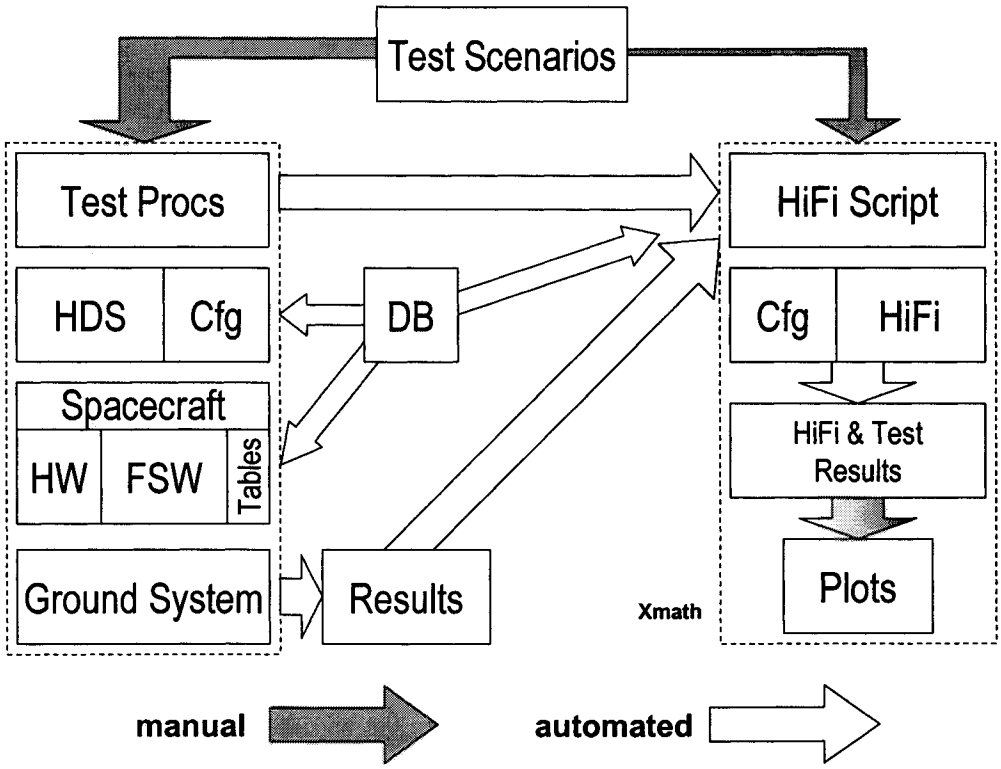

Figure 3: Automating Testing Steps Makes Test Process More Efficient

The different pieces of the test process flow shown in Figure 3 will be discussed in the next two sections, along with the software tools that implement them.

INTEGRATED DESIGN ENVIRONMENT

In order to be able to apply the automated test tools developed for the MAP flight software test effort to maximum effect, it was necessary to combine the elements of the subsystem and the testing tools into an integrated design and testing environment.

Parameter Database

One of the key elements to the success of the MAP flight software testing effort was the parameter database used by the ACS subsystem. This database was used to configuration manage virtually all of the variables, control gains, failure detection and correction (FDC) parameters, and other parameters used by the MAP ACS. As shown in Figure 3, the database fed into all of the ACS elements. As each parameter was placed in the MAP database, it was assigned to an appropriate subsystem engineer, in order for the database to be populated with the correct information and verified.

Upon a release each new version of flight software for either the spacecraft main processor or attitude control electronics, a corresponding release from the MAP database was created. Output templates from the database were created as header files for the flight software; additionally, script files for initializing the HDS and HiFi were generated at the same time. In this way, when flight software tests and HiFi verification simulations were performed, a consistent set of parameters across each test system component was assured.

Scripting Language with String Processing

The MathScript scripting language of MatrixX was used extensively with the MAP HiFi in order to set up simulations and to perform many data analysis functions. MathScript allows flight software and HiFi simulation to be analyzed, it interfaces with MatrixX's SystemBuild simulation environment to allow simulations to be created and run, and provides the mechanism for creating comparison plots between flight software and HiFi simulation verification data.

MathScript is a very complete scripting language for data processing, particularly matrix processing, and provides all of the functions necessary for interfacing numerical data and the HiFi simulation. One capability it lacks, however, is for doing extensive string processing. Because of the need to process the sequentially printed data output files from the MAP ground system, which are mixed numeric and text, and to process STOL procedures and RDLs (which are used to run test procedures and describe MAP data packets, respectively) as well, string processing is necessary.

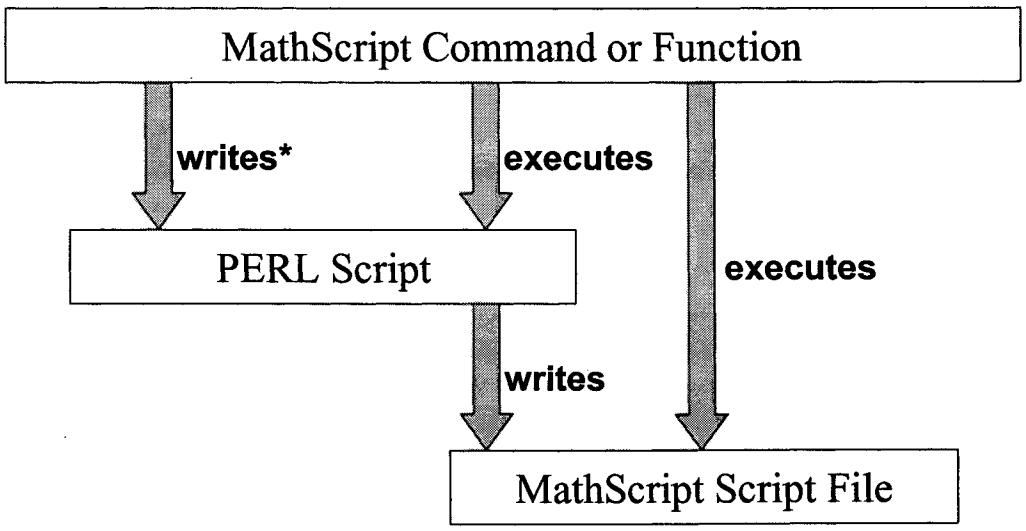

Figure 4: MathScript Architecture Allows Additional Capabilities to be Added

Figure 4 shows the solution implemented to add the string processing abilities necessary for the MAP test tools. The solution makes use of the ability that MathScript has to call external commands native to the underlying operating system. In order to add string processing to MathScript functions, the following steps were followed:

1. A MathScript function, using its `fprintf` statements, writes a PERL script out to the file system. (In some cases, a pre-existing PERL script is used instead, in which case this step is not necessary.)

2. Using a native operating system command, the PERL script from step 1 is executed.

3. The PERL script executes and processes the test procedure sequential print data output files, STOL procedures, or RDLs, as appropriate. As output, the PERL script writes out a new MathScript script file.

4. Finally, the original MathScript function calls the newly created script file, which might, for example, read the test data into MatrixX.

When the MAP test tools were originally developed, a combination of the Unix commands and utilities of bash, sed, and grep were used to implement these functions. Eventually, though, PERL was used instead because it could perform all of the functions that were divided between the other utilities, plus it is widely available on many different platforms.

Configured HiFi Simulation

Figure 5 shows the top-level block diagram of the MAP HiFi simulation, shown here to give a sense of what the simulation looks like as implemented in the MatrixX SystemBuild simulation environment. SystemBuild is a hierarchical block diagram editor made up of SuperBlocks, such as the ones shown in Figure 5, and lower level blocks that can be used to implement the different mathematical functions and dynamic elements needed in a simulation.

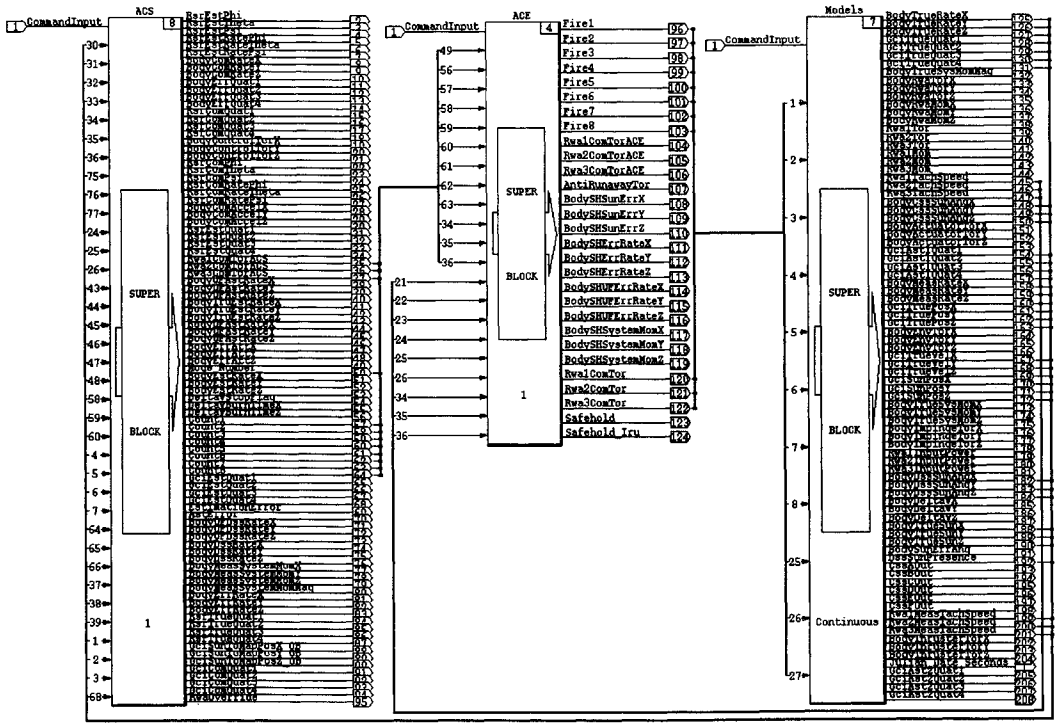

Figure 5: MatrixX Simulation Environment Lends Itself to Automation

The MatrixX integrated toolkit, its Xmath command environment (in which the MathScript scripting language is based), and its SystemBuild simulation component lend themselves especially well to the automation process devised in the MAP test tools. As described in the previous section, the scripting language acts as the glue that holds the simulation and testing process together. Through the string processing extensions, test data can be read in and processed. Using SystemBuild Access, an extension to Xmath that allows it to access and control SystemBuild simulations, MathScript functions can complete the process of data analysis and test verification by creating and running HiFi simulations.

It should be noted that while the MatrixX integrated toolkit lends itself particularly well to the design of automated test tools, such tools could be designed in other settings. The key ingredient that is necessary is a scripting language around which the rest of the system can be built. The Matlab/Simulink environment, using Matlab's m-file scripting, could also be used. Even a dedicated simulation such as one written in a language such as FORTRAN or C can be used with the automated test techniques described herein by using an environment such as Matlab or MatrixX as a front end interface.

Plotting Capability

The final capability needed to support the automated test tools is a flexible plotting capability. By being able to plot test data and HiFi verification simulation data on a common plot, the verification process is made much simpler. Along with the test tools, which will be described in the next section, that input test data and automatically set up HiFi simulations, the plotting of an arbitrary number of verification plots is supported. This becomes the final step in the test verification process; after comparison plots are created, they can be analyzed to verify a flight software test.

TOOL DESCRIPTIONS

In this section, the specific automated test tools that were developed for use on the ACS flight software testing for MAP will be discussed. These are the MathScript and PERL functions that implement the capabilities described in the previous section.

Test Data Processing

The main tool for processing the sequential print data output from flight software testing is called `pktproc`. This procedure, using the PERL-enhanced string processing methods described in the previous section, reads in STOL sequential print data and description files. It is able to automatically combine split packets (large packets must sometimes be split to accommodate ground system limitations). Each packet of data is formatted into an Xmath PDM (a data structure used by Xmath that is able to store a matrix along with auxiliary data, such as a time domain vector, and names applied to each column of data, in this case the packet telemetry names). The `pktproc` routine then writes out each set of packet data PDM for later processing by other Xmath test tools.

There are other test data processing functions used by the MAP test tools. These are used for post-processing flight software test data, calculating parameters not downlinked directly.

Synchronization and Data Analysis

Once the flight software test data has been processed and imported into the Xmath environment, the two most important functions of the automated test tools are implemented. The first is implemented with the `msync` function. This function finds the beginning of a test run and synchronizes the test data from each packet. This is necessary because when sequential prints are started at the beginning of a flight software test, each packet begins to be output at a different time. A standard test format was established for the MAP flight software testers in which all of the sequential prints were begun, and then the "restart" process was begun in the FlatSat flight software test facility which set the initial conditions for the test. The `msync` function analyzes the appropriate data packet for signs in the output data of this "restart". It then determines the offset into each packet that corresponds to this time. The offsets are saved for use by the other test tools. In this way, though the packets each begin at a different time, the "zero point" for each packet in the test is determined.

In addition to the `msync` function, the other critical test data analysis function for purposes of implementing the MAP automated test tools is the `transitions` function. The `transitions` function examines many of the different test data output packets for different aspects of the test run. This information is then used to *automatically* create a MathScript script file for a HiFi simulation that will match the test run. Some of the flight software test conditions that `transitions` looks for are the following:

- initial conditions: time, spacecraft position and velocity, initial spacecraft attitude quaternion, system momentum, and body rates
- Safehold and Safehold rate sensor transitions
- ACS control mode transitions
- Observing Mode spin down transitions
- thruster one-shots
- command quaternion transitions
- reaction wheel override commands
- Delta V Mode commanded burn times
- commands to enable or disable the Delta V impulse controller
- HDS commands to enable or disable sensor and actuator noise

In many ways, the `transitions` function is the heart and soul of the MAP automated test tools. By allowing software testers to automatically create matching HiFi simulations, it is possible for anyone, not simply ACS analysts familiar with the HiFi, to create HiFi verification runs for flight software tests. Analysts are still necessary for verifying the flight software tests,

but by making the creation of the HiFi verification run automatic, one potential bottleneck is removed from the test and test verification process.

Plotting and Test Verification

Once the flight software test data is imported into Xmath and analyzed, and a HiFi verification simulation run, the final step is to plot the test data along with the HiFi data for comparison. Because of the data synchronization and automatic HiFi simulation setup, it is a simple matter to create these comparison plots.

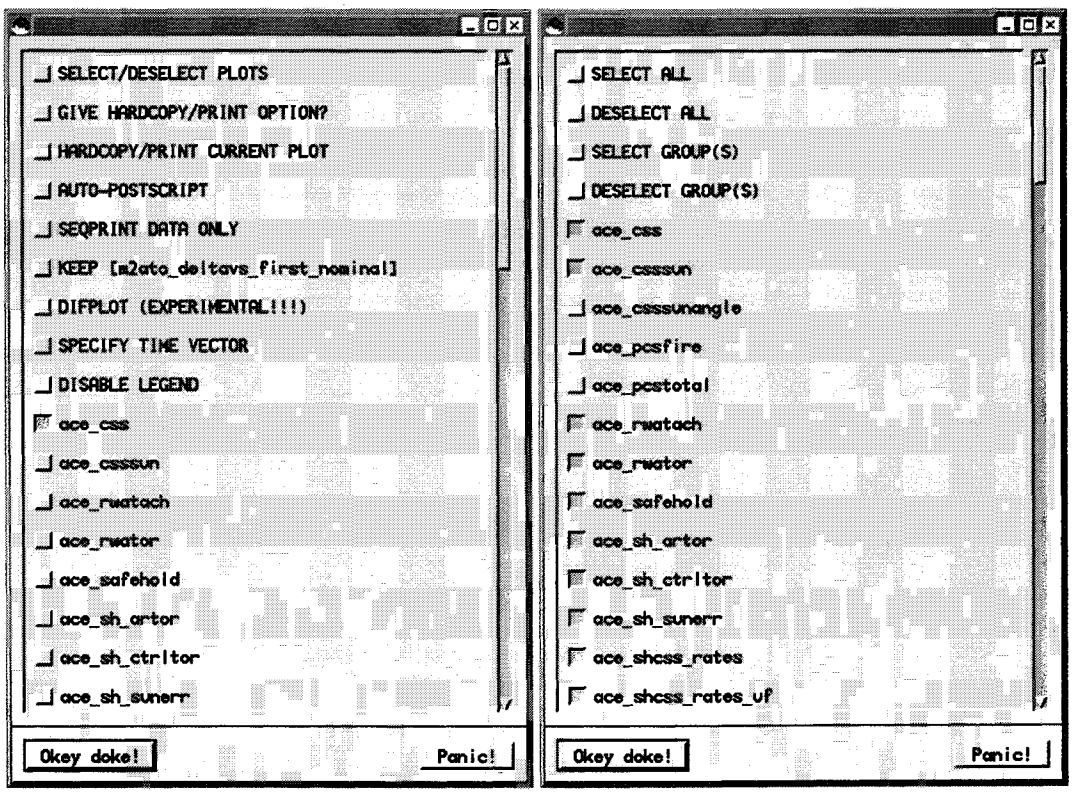

Figure 6: Plot Tools Offer Flexibility and Batch Processing

Figure 6 shows the graphical interface to the test tool `mapplot` function. This function gives the user the ability to create one or more of many predefined comparison plots. If a plot is selected, then test data and HiFi data are plotted together, such as in the plot shown in Figure 7. As shown on the left of Figure 6, there are a number of options that the user can select in addition to picking one of the plots. If the `SELECT/DESELECT PLOTS` option is selected, the window shown at right is given, which allows the user to select which plots are appropriate for a given run. Once back at the main screen, any of these plots can be created and, if desired, copied to a printer or a PostScript file. In addition, the `AUTO-POSTSCRIPT` option allows for all of the selected plots to be automatically created and copied to PostScript files.

The `mapplot` function included a number of globally defined plots that users could employ. By defining a standard set of plots, it became that much easier for each tester to produce a complete review package for each test run. In addition, mapplot supported a very simple format

for each user to define their own plots, either globally across all of their flight software tests, or specific to a given test.

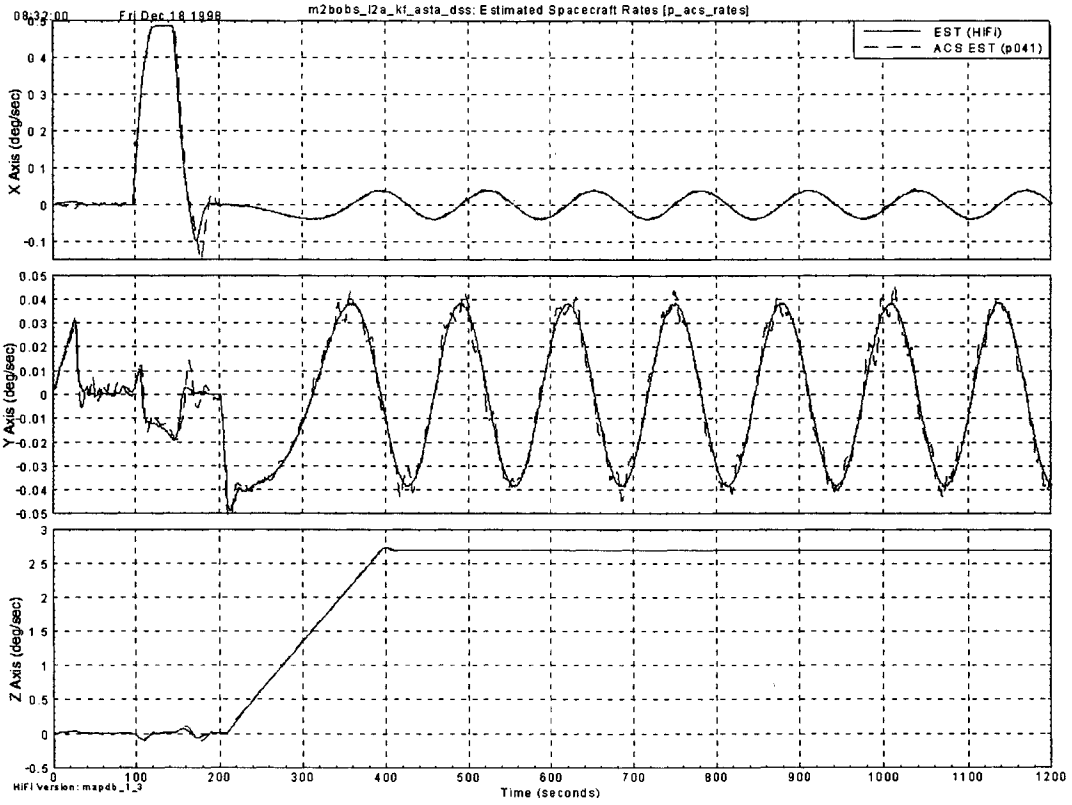

Figure 7: Synchronized Data on Common Plots Aids Test Verification

In addition to the `mapplot` function, a lower-level plotting function was provided in the test tools, called `pktplot`. Unlike `mapplot`, which provided a choice of a number of predefined plots, `pktplot` gave access to any telemetry point in any test data output packet, or any HiFi output variable. This ability was primarily meant for troubleshooting problems or anomalies in a given flight software test, or for producing a very specific plot.

One final piece of the plotting capabilities of the MAP automated test tools should be mentioned. In defining the plots used by the `mapplot` function, the tools only make it necessary to specify the telemetry point desired. How then do the tools know what data packet a given telemetry point is in? This information is kept in the "packet key". A `setpktkey` function was written which operated in the background, calling a PERL script to process through the standard set of sequential print STOL procedures that all flight software tests employed. By parsing out the telemetry points from each STOL procedure, a packet key matching telemetry points and packets was established. This packet key was then used by the other test tools to figure out where to find each needed telemetry point.

Summary of Flight Software Test Verification Steps

Referring back to Figure 3, which detailed the automated test process used on the MAP project, the following short list of steps reflect everything needed to be done to perform a flight software test and its verification:

1. From a flight software test scenario, write a STOL procedure to test that scenario on FlatSat.
2. Copy the sequential print output files to the MatrixX computer.
3. Within MatrixX, run the `pktproc` function to process the data; `pktproc` automatically calls `msync` and `transitions` to synchronize the data and set up the HiFi simulation.
4. After a review of the HiFi simulation script created in the previous step, run the simulation.
5. Use the `mapplot` function to create an appropriate set of comparison plots.
6. Review the comparison plots to determine whether or not the flight software test passed.

As can be seen in the above process, the only major manual step is in step 1, where the original flight software test procedure is created. Other than possible manual tweaks that might be required for the HiFi verification simulation in step 4, everything else is automated by the MAP test tools until the final step where the flight software test data is compared with the data from the HiFi simulation and a PASS or FAIL is given to the test.

REUSABILITY

The flight software test tools used for MAP ACS flight software testing and described above were not the result of a formal development process. Rather, the tools began as an assortment of testing shortcuts from one of the MAP testers; as the collection grew, these shortcuts were assembled into a toolset that eventually allowed much of the test verification process to be automated. Because the tools were written in an assortment of MatrixX's MathScript scripting language and the Unix utilities and languages of bash, grep, sed, and PERL, and not thoroughly documented, they were exclusively geared towards use on the MAP project.

When development of the Triana spacecraft began at Goddard, the Triana ACS subsystem team chose to base their high fidelity simulation on MAP's. Because this meant that the Triana HiFi was closely related to the MAP HiFi, it was logical to assume that Triana could also benefit from the test tools developed for MAP, after they were converted for Triana's use.

Tool Conversion

The decision was made to reuse the MAP test tools after the Triana lead ACS analyst advocated their use. The MAP test tools were demonstrated for the flight software test lead, the analytical team and the ACS flight software lead. Everyone was enthused, and agreed that their use would be beneficial for Triana. Input was solicited on what people wanted to keep and what they didn't want, and based on this input, Dave McComas, a member of the MAP flight software development team, worked on reshaping the tools to Triana's wants and needs. While not the

original developer of the tools, he was familiar enough with MAP, MatrixX, and the Unix utilities to be a good candidate for adapting them for Triana.

The Triana team identified the following features of the MAP test tools as the ones they would be most interested in:

1. the ability to time synchronize test data
2. the ability to plot data, and to batch produce a set of plots
3. the ability to automatically make a matching HiFi verification simulation

Further, due to time and resource constraints, Triana chose to adopt only the first and second features shown. In terms of the specific MAP test tools, Triana versions of `pktproc`, `mapplot`, `msync`, and `pktplot` were developed. The routine used to automatically produce a HiFi script corresponding to the test, `transitions`, was not adapted.

Because of the nature of the MAP test tools, the first step necessary before adapting them to use on Triana was to "reverse-engineer" and document them, learning how they work. This was done with the assistance of the original tool developer. The tools and techniques themselves were not difficult to translate, once the underlying assumptions about standards were understood. As mentioned, a big reason that the MAP test tools worked as well as they did was because of the widespread use of standard test procedure and data output formats established for MAP testing. Once the assumptions about these standards were understood, it became possible to adapt them to Triana by establishing corresponding standards there.

The greatest difficulties in translating the MAP test tools for Triana was caused by differences in the ground systems used on the two projects. In particular, the Triana system worked with different time formats; in order to be able to synchronize data it is vital to be able to work with compatible times, so establishing a compatible standard was important. Also, because the Triana ground system had to deal with data rates much higher than with MAP (10 Hz vs 1 Hz data) and would sometimes drop data packets more frequently, the Triana test tools needed to be able to deal with that.

After discussions with the Triana test team and a number of iterations, a workable set of tools was produced for use on that project. Additionally, unlike with the "home grown" set of MAP test tools, a complete user's guide was produced.

Experience with the Triana Test Tools

The experience of using the automated test tools on Triana ended up being significantly different than MAP. In spite of the early enthusiasm on the part of the ACS flight software and flight software test leads, as well as the ACS analysis team, the entire group did not "buy in" to the use of the tools as much as would have been desired.

On the positive side, there were some features of the automated test tools that everyone liked, and the tools saw their most use for these things. Especially useful was the tools' ability to synchronize test data from different packets to allow it to be easily "lined up" within a data plot.

The ability to batch output an arbitrarily long series of standardized plots for a test data run was also found to be useful. Finally, the point and click user interface to plotting (as shown above in Figure 6) was liked.

For Triana, there ended up being more negative feelings toward the test tools than positives for many members of the test team. Because of the lack of integration of the flight software and ACS teams, there was a greater division of labor than was present on MAP. Combined with the fact that the `transitions` procedure used to "close the loop", automatically producing a HiFi script to correspond to a given test data set, was not adapted for Triana, this meant that there was much less perceived benefit from using the tools. Further, because many members of the team were not familiar with MatrixX or Unix, and were not provided with the tools until after the project was already underway, they were less likely to want to change the way they were doing things in order to use them.

In summary, the data synchronization and batch plotting abilities of the test tools were found to be invaluable to the Triana project. However, because of the less-integrated flight software test and ACS teams on Triana, the "imposed" nature of the tool use resulting in less buy in from the team, and the fact that many members of the team had established their own procedures before the tools became available, meant that the automated test tools did not make as big an impact on the Triana project as on MAP.

CONCLUSION

What began as an *ad hoc* development of software "shortcuts" for ACS flight software testing on the MAP spacecraft evolved over time into a very useful set of tools for greatly increasing the efficiency of the testing process. By standardizing interfaces and test procedures and developing new data analysis tools, the MAP project was able to leverage its existing ACS subsystem-wide parameter database, an integrated test team consisting of flight software developers, testers, and ACS analysts, and the MatrixX integrated simulation, analysis, and plotting toolkit, to remove many of the testing bottlenecks that slowed down projects in the past. As a result, MAP was able to go through build and acceptance testing, as well as a number of rounds of regression testing to accommodate late software changes and additions, in a very timely, efficient, and thorough manner.

Because the HiFi simulation used for the Triana project also used MatrixX and was directly based on the MAP HiFi, it would have seemed that Triana was an ideal candidate for also making use of the MAP test tools. However, the experience of trying to apply them for use on Triana showed some of the potential pitfalls and limitations when attempting this sort of reuse. Because the MAP test tools were not the result of a formal development process, they were not well-documented at first. It took some time for the conversion and documentation process to be performed, which meant that the tools were not available to Triana from the beginning. Additionally, on many levels the Triana project was not as integrated as MAP, with no central parameter database for all components of the testing environment, separate ACS and flight software test teams, and more difficulty establishing the standard test procedure and data output formats upon which the MAP tools depend. Nevertheless, while not used as widely or universally on Triana as on MAP, the tools did provide some benefit to the Triana's ACS flight

software testing effort. It is clear that the techniques inherent in the MAP test tools, if not the actual tools themselves, can be applied to great benefit in future projects.

REFERENCES

[1] McComas, David C., James R. O'Donnell, Jr., Ph.D., and Stephen F. Andrews, "Using Automatic Code Generation in the Attitude Control Flight Software Engineering Process", *23rd Software Engineering Workshop*, NASA Goddard Space Flight Center, Greenbelt, MD, 1998.

[2] Ward, David K., Stephen F. Andrews, David C. McComas, and James R. O'Donnell, Jr., Ph.D., "Use of the MatrixX Integrated Toolkit on the Microwave Anisotropy Probe Attitude Control System," *21st AAS Guidance and Control Conference*, Breckenridge, CO, 1999.

[3] O'Donnell, James R., Jr., Ph.D., Stephen F. Andrews, David C. McComas, and David K. Ward, "Development and Testing of Automatically-Generated Flight Software for the MAP Spacecraft," *14th International Symposium on Space Flight Dynamics*, Iguassu Falls, Brazil, 1999.